

```

<book>
  <title> XML </title>
  <allauthors>
    <author> jane </author>
    <author> john </author>
  </allauthors>
  <year> 2000 </year>
  <chapter>
    <head> Origins </head>
    <section>
      <head> ...</head>
      <section> ...</section>
    </section>
    <section> ...</section>
  </chapter>
  <chapter> ...</chapter>
</book>
    
```

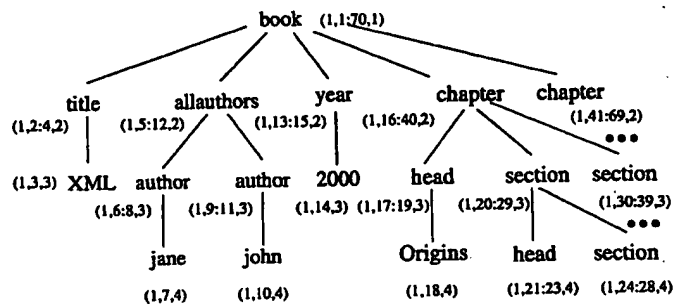


FIG. 1A

FIG. 1B

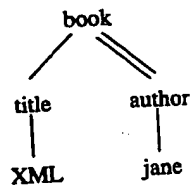


FIG. 2A

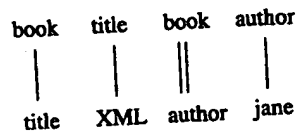


FIG. 2B

```
Algorithm Tree-Merge-Anc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* AList is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descendants in sorted order of StartPos */

begin-desc = DList->firstNode; OutputList = NULL;
for (a = AList->firstNode; a != NULL; a = a->nextNode) {
    for (d = begin-desc; (d != NULL && d.StartPos < a.StartPos); d = d->nextNode) {
        /* skipping over unmatchable d's */
    }
    begin-desc = d;
    for (d = begin-desc; (d != NULL && d.EndPos < a.EndPos); d = d->nextNode) {
        if ((a.StartPos < d.StartPos) && (d.EndPos < a.EndPos)
            && (d.LevelNum = a.LevelNum + 1)) {
            /* the optional condition is for parent-child relationships */
            append (a,d) to OutputList;
        }
    }
}
```

FIG. 3

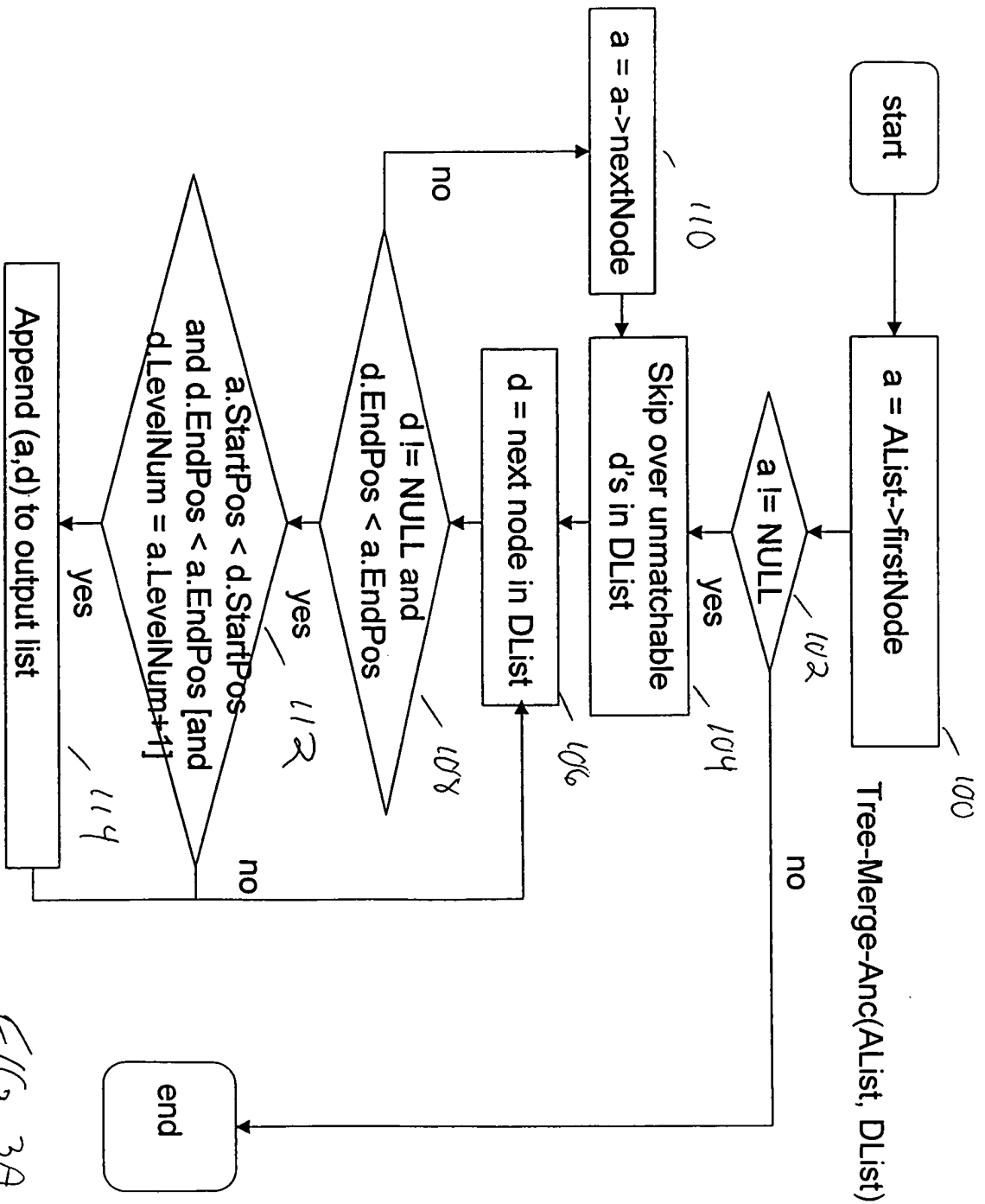
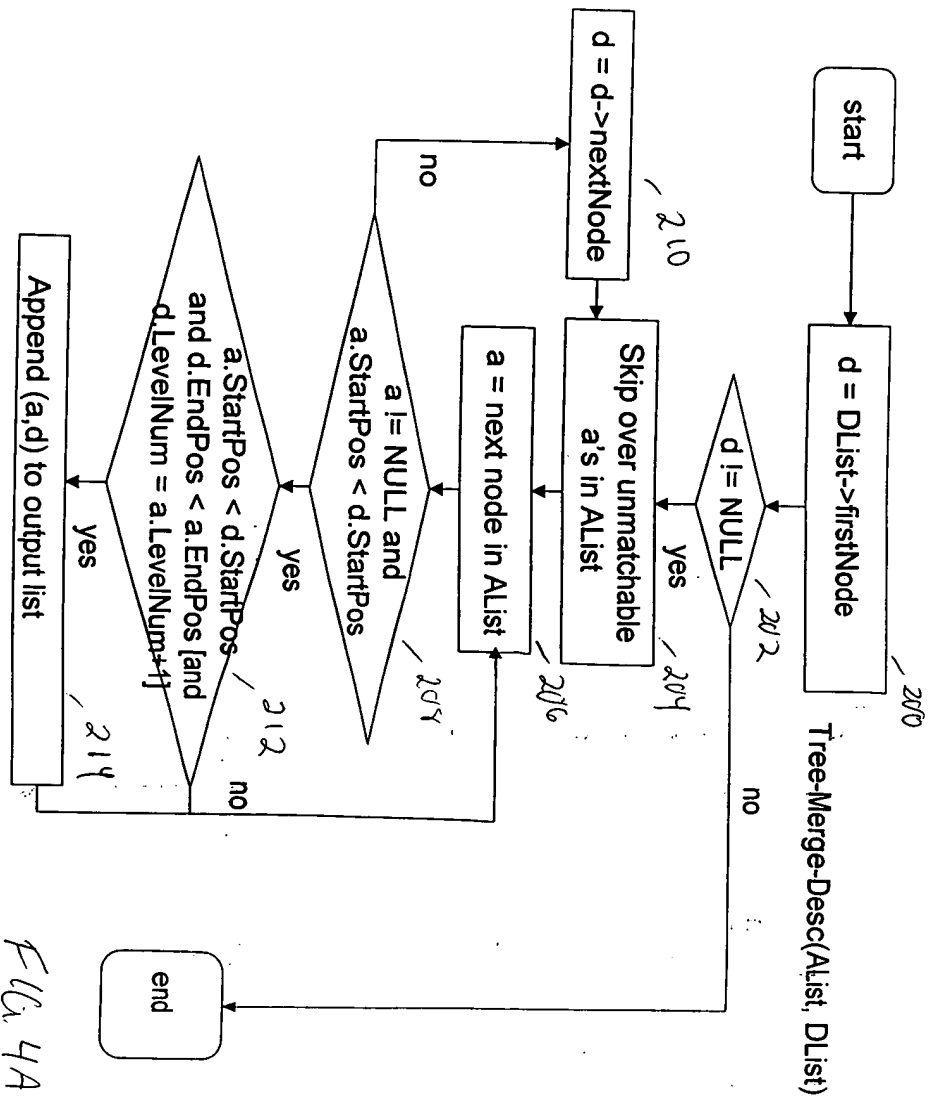


FIG. 3A

```
Algorithm Tree-Merge-Desc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* AList is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descendants in sorted order of StartPos */

begin-anc = AList->firstNode; OutputList = NULL;
for (d = DList->firstNode; d != NULL; d = d->nextNode) {
  for (a = begin-anc; (a != NULL && a.EndPos < d.StartPos); a = a->nextNode) {
    /* skipping over unmatchable a's */ }
  begin-anc = a;
  for (a = begin-anc; (a != NULL && a.StartPos < a.StartPos); a = a->nextNode) {
    if ((a.StartPos < d.StartPos) && (d.EndPos < a.EndPos)
        && (d.LevelNum = a.LevelNum + 1)) {
      /* the optional condition is for parent-child relationships */
      append (a,d) to OutputList; }
  }
}
```

FIG. 4



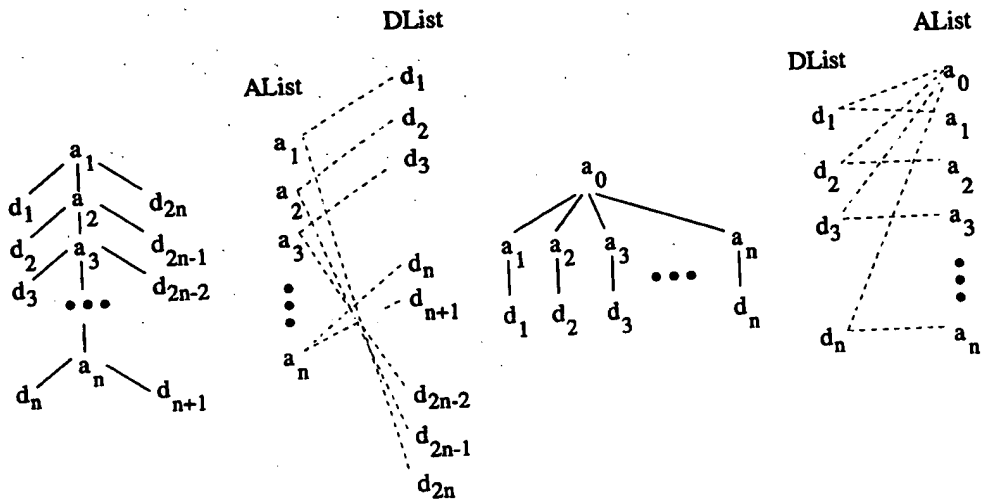


FIG. SA

FIG. SB

FIG. SC

FIG. SD

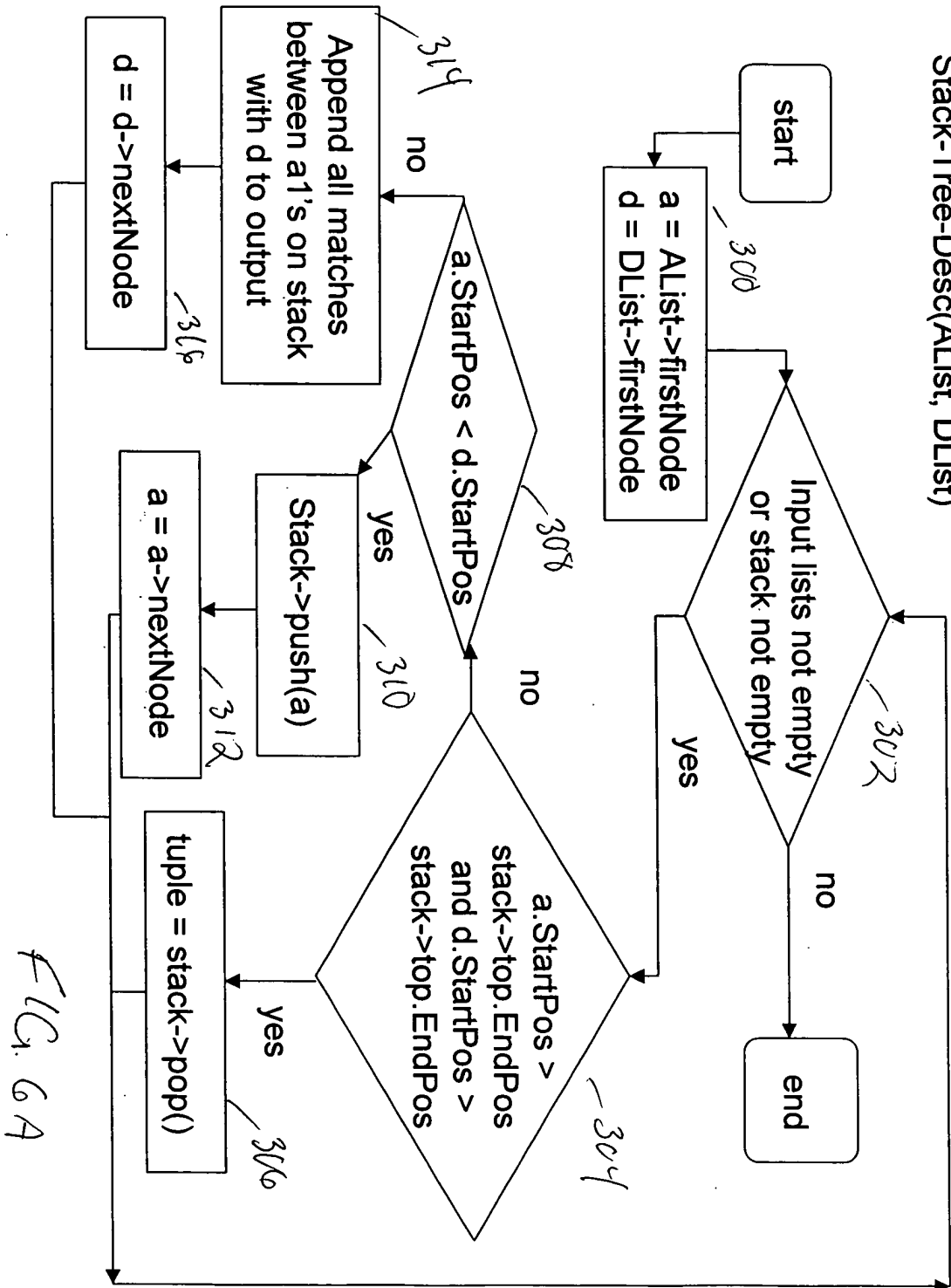
```
Algorithm Stack-Tree-Desc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* AList is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descendants in sorted order of StartPos */

a = AList->firstNode; d = DList->firstNode; OutputList = NULL;
while (the input lists are not empty or the stack is not empty) {
    if ((a.StartPos > stack->top.EndPos) && (d.StartPos > stack->top.EndPos)) {
        /* time to pop the top element in the stack */
        tuple = stack->pop(); }
    else if (a.StartPos < d.StartPos) {
        stack->push(a)
        a = a->nextNode }
    else {
        for (a1 = stack->bottom; a1 != NULL; a1 = a1->up) {
            append (a1,d) to OutputList
        }
        d = d->nextNode
    }
}
```

FIG. 6

9/16

Stack-Tree-Desc(AList, DList)



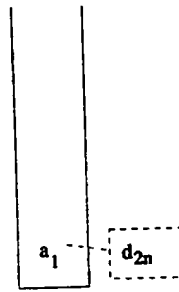
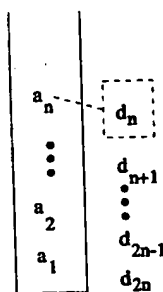
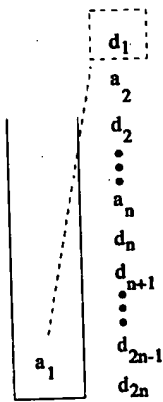
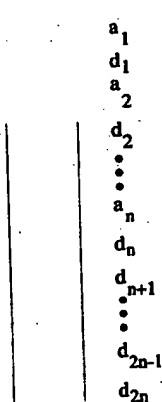
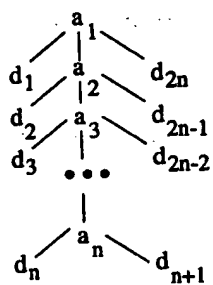


FIG. 7A

FIG. 7B

FIG. 7C

FIG. 7D

FIG. 7E

```
Algorithm Stack-Tree-Anc (AList, DList)
/* Assume that all nodes in AList and DList have the same DocId */
/* AList is the list of potential ancestors, in sorted order of StartPos */
/* DList is the list of potential descendants in sorted order of StartPos */

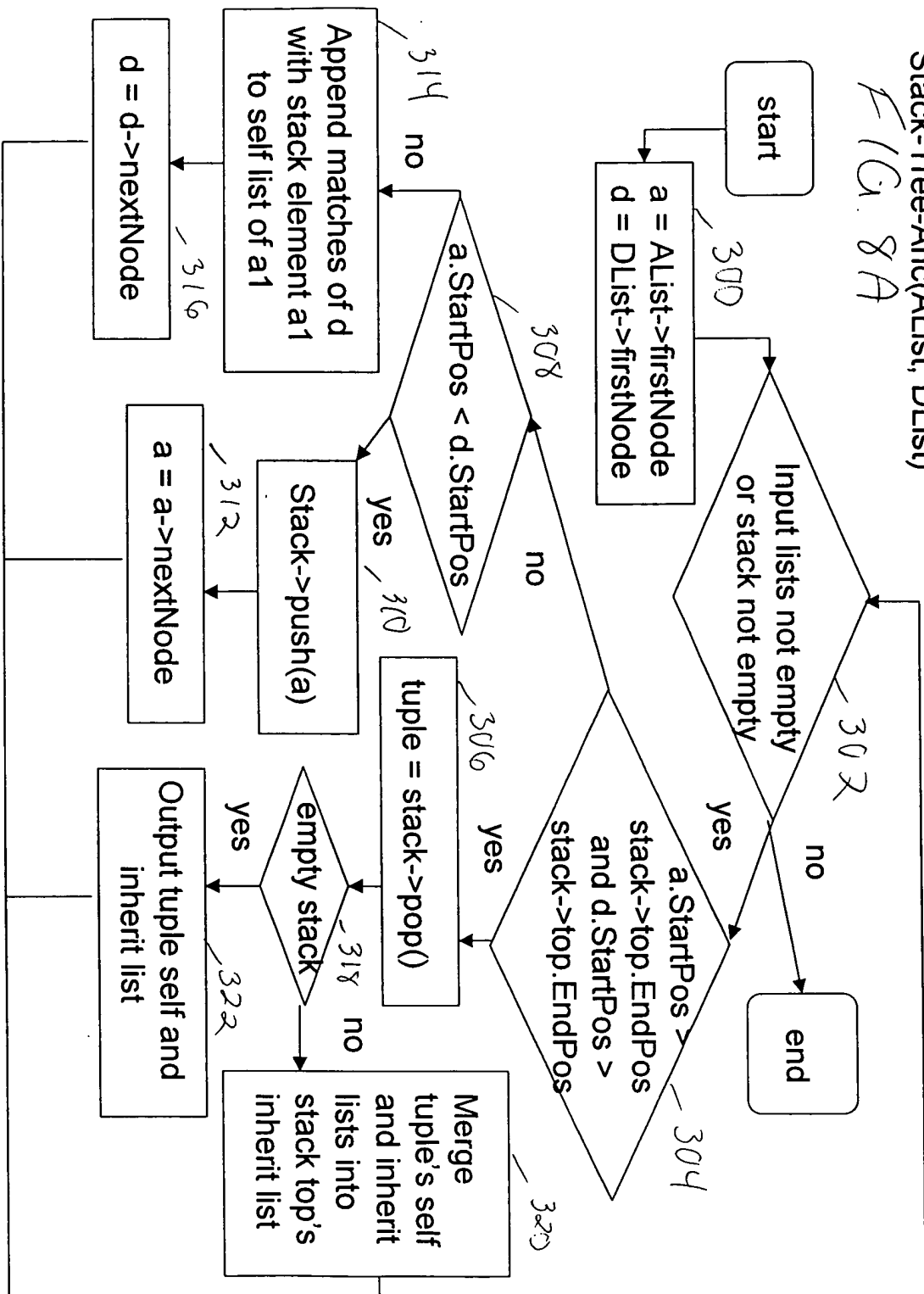
a = AList->firstNode; d = DList->firstNode; OutputList = NULL;
while (the input lists are not empty or the stack is not empty) {
  if ((a.StartPos > stack->top.EndPos) && (d.StartPos > stack->top.EndPos)) {
    /* time to pop the top element in the stack */
    tuple = stack->pop();
    if (stack->size == 0) { /* we just popped the bottom element */
      append tuple.inherit-list to OutputList
    }
    else {
      append tuple.inherit-list to tuple.self-list
      append the resulting tuple.self-list to stack->top.inherit-list
    }
  }
  else if (a.StartPos < d.StartPos) {
    stack->push(a)
    a = a->nextNode
  }
  else {
    for (al = stack->bottom; al != NULL; al = al->up) {
      if (al == stack->bottom) append (al,d) to OutputList
      else append (al,d) to the self-list of al
    }
    d = d->nextNode
  }
}
```

FIG. 8

12/16

Stack-Tree-Anc(AList, DList)

FIG. 8A



```
<!ELEMENT manager (name, (manager|department|employee)+)>  
<!--ATTLIST manager id CDATA #FIXED "1"-->  
<!ELEMENT department (name, email?, employee+, department*)>  
<!--ATTLIST department id CDATA #FIXED "2"-->  
<!ELEMENT employee (name+, email?)>  
<!--ATTLIST employee id CDATA #FIXED "3"-->  
<!ELEMENT name (#PCDATA)>  
<!--ATTLIST name id CDATA #FIXED "4"-->  
<!ELEMENT email (#PCDATA)>  
<!--ATTLIST email id CDATA #FIXED "5"-->
```

FIG. 9

METHOD OF PATTERN SEARCHING
Nikolaos Koudas, et al.
ATT-105AUS

14/16

Node	Count
manager	25,880
department	342,450
employee	574,530
email	250,530

Query	XQuery Path Expression	Result Cardinality
QS1	employee/email	140,700
QS2	employee//email	142,958
QS3	manager/department	16,855
QS4	manager//department	587,137
QS5	manager/employee	17,259
QS6	manager//employee	990,774
QC1	manager/employee/email	7,990
QC2	manager//employee/email	232,406

FIG 9A

FIG 9B

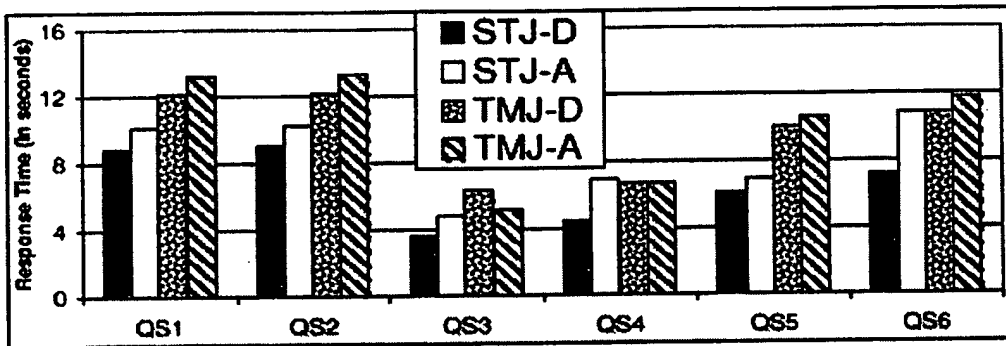


FIG. 10

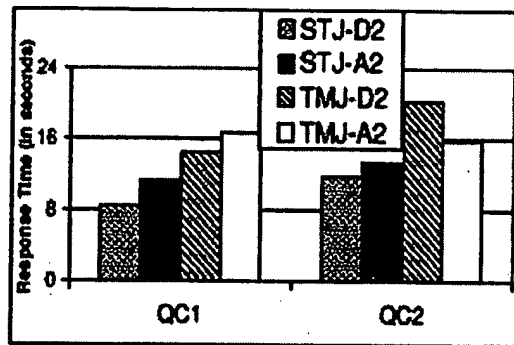


FIG. 11